**Лабораторная работа №7**

**“Функции”**

При выполнении лабораторной работы используйте процедуры и функции:

1. Функция чтения

2. Функция обработки (может быть несколько)

3. Функция записи

Их необходимо расположить в отдельных файлах (\*.hpp и \*.cpp) в отдельном пространстве имен.

Вариант 1

Дана целочисленная матрица {Aij}i=1...n;j=1..n , n<=100. Если в матрице есть два одинаковых столбца и есть хотя бы один элемент, абсолютная величина которого - простое число, упорядочить строки матрицы по неубыванию суммы модулей элементов.

Вариант 2

Дана целочисленная матрица {Aij}i=1...n;j=1..n , n<=100. Если в матрице есть две одинаковых строки, заменить минимальные элементы столбцов на наибольший из простых элементов матрицы.

Вариант 3

Дана целочисленная матрица {Aij}i=1...n;j=1..n , n<=100. Если все диагональные элементы матрицы положительны и содержат цифры 3 или 5, заменить минимальные элементы столбцов на суммы квадратов элементов соответствующих столбцов.

Вариант 4

Дана целочисленная матрица {Aij}i=1...n;j=1..n , n<=100. Если в матрице есть еще один элемент, равный ее минимальному элементу, и не менее 2-х элементов, абсолютные величины которых - простые числа, упорядочить строки матрицы по невозрастанию произведений элементов.

Вариант 5

Дана целочисленная матрица {Aij}i=1...n;j=1..n , n<=100. Если в матрице есть еще один элемент, равный ее максимальному элементу, упорядочить строки матрицы по невозрастанию количества простых чисел среди элементов строк.

Вариант 6

Дана целочисленная матрица {Aij}i=1...n;j=1..n , n<=100. Если все диагональные элементы матрицы являются наибольшими элементами своих строк, заменить элементы матрицы, содержащие цифру 0, на произведение диагональных элементов.

Вариант 7

Дана целочисленная матрица {Aij}i=1...n;j=1..n , n<=100. Если суммы цифр минимального и максимального элементов матрицы одинаковы, упорядочить строки матрицы по неубыванию суммы элементов.

Вариант 8

Дана целочисленная матрица {Aij}i=1...n;j=1..n , n<=100. Если суммы цифр минимального и максимального элементов матрицы отличаются не более, чем на 2, упорядочить столбцы матрицы по неубыванию суммы элементов.

Вариант 9

Дана целочисленная матрица {Aij}i=1...n;j=1..n , n<=100. Если сумма наибольшего и наименьшего элементов матрицы равна нулю, упорядочить элементы строк, в которых есть хотя бы один элемент, содержащий цифру 8, по неубыванию.

Вариант 10

Дана целочисленная матрица {Aij}i=1...n;j=1..n , n<=100. Если в матрице есть столбец, не содержащий простых чисел, сумма элементов которого равна 0, упорядочить элементы строк по неубыванию абсолютных величин.

Header файлы .hpp и.h .

Они нужны, чтобы разбить код на части и постоянно не компилировать всё.

func.hpp:

#pragma once

namespace func{

int sum(int a, int b); //реализация не здесь

}

func.cpp:

namespace func{

int sum(int a, int b){

return a + b

} // реализация здесь

}

main.cpp:

#include <iostream>

#include “func.hpp”

int main(){

cout << func::sum(2,3);

return 0;

}

//#include “func.hpp” “” - так будет искать header в папке рядом с main.cpp, скобки типо <> обычно ищут среди системных папок.. но если указать в cmake include\_directories(“fold\_name/”)

По факту в main cpp нет реализации sum, ровно как и в func.hpp. .hpp просто говорит main.cpp, что где-то эта реализация есть. Поэтому нам не нужно постоянно компилировать эту функцию sum. Например мы скомпилировали func.cpp и main.cpp один раз, но потом поменяли main cpp. Запускаем компилятор: func.cpp уже скомпилировалась до этого, не перекомпилируем, скомпилировалась main.cpp с указанием на то, что где-то есть реализация sum. Потом, происходит линковка и уже там main.cpp в func.cpp находит реализацию sum. Программа скомпилировалась и будет работать.

Поэтому нет смысла в полностью header библиотеках(реализация функций находится в header) в плане скорости, он будет перекомпилироваться при каждом изменении.